Daniel Marshall
300440531

COMP307
Assignment 1

16/03/2021

Assignment 1: Basic Machine Learning Algorithms

Part 1: k-Nearest Neighbours Method

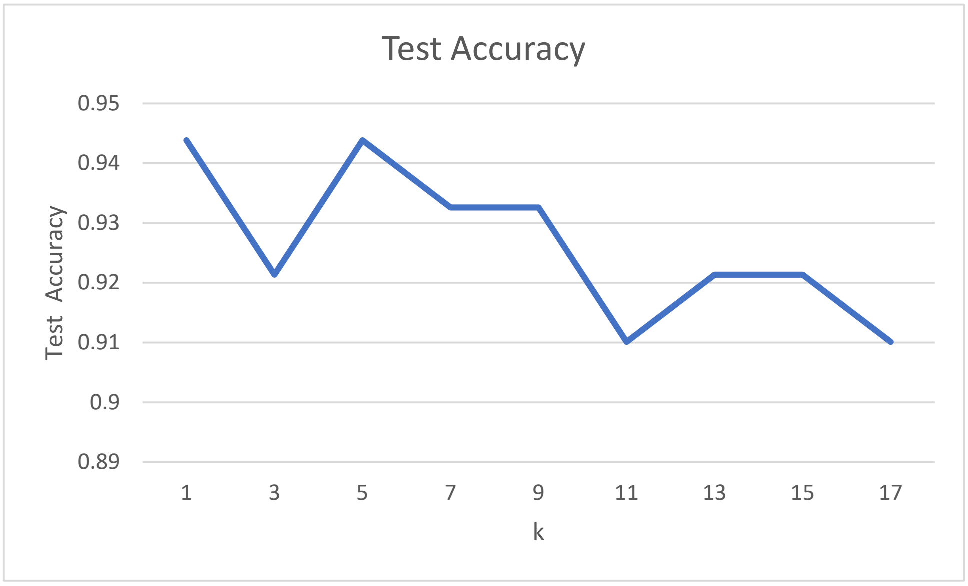
1. Report the class labels of each instance in the test set predicted by the basic nearest neighbour method (where k=1), and the classification accuracy on the test set of the basic nearest neighbour method. Make sure you keep the same order as in the test set file.

Row Index	Predicted Class	Actual Class	Row Index	Predicted Class	Actual Class	Row Index	Predicted Class	Actual Class
1	3	3	34	1	1	67	3	3
2	3	3	35	2	2	68	1	1
3	3	3	36	2	2	69	1	1
4	1	1	37	3	2	70	2	2
5	1	1	38	2	2	71	2	2
6	1	1	39	2	2	72	3	3
7	2	2	40	1	1	73	2	2
8	1	2	41	2	2	74	2	2
9	1	1	42	2	2	75	1	1
10	2	2	43	3	3	76	1	1
11	3	2	44	1	1	77	1	1
12	3	2	45	2	2	78	3	3
13	3	3	46	1	1	79	1	1
14	3	3	47	3	3	80	1	1
15	1	1	48	2	2	81	2	2
16	2	2	49	2	2	82	2	2
17	3	3	50	1	1	83	3	3
18	3	3	51	3	3	84	1	1
19	1	1	52	1	1	85	2	2
20	1	1	53	1	1	86	1	1
21	3	3	54	3	3	87	1	1
22	2	2	55	3	3	88	2	2

23	2	2	56	1	1	89	1	1
24	3	3	57	1	1			
25	2	2	58	3	3			
26	3	2	59	1	1			
27	2	2	60	3	3			
28	3	3	61	3	3			
29	2	2	62	1	2			
30	1	1	63	2	2			
31	2	2	64	3	3			
32	1	1	65	2	2			
33	2	2	66	3	3			

```
84/89 instances classified correctly
Accuracy: 94.38%
```

2. Report the classification accuracy on the test set of the k-nearest neighbour method where k=3 and compare and comment on the performance of the two classifiers (k=1 and k=3).



When k=1, a total of 84/89 instances are classified correctly, an accuracy of 94.38%. When k=3 however, a total of 82/89 instances are classified correctly, an accuracy of only 92.13%. This is still very good, and quite close to k=1, so I believe the difference to not be statistically significant. To see whether this was a trend, I plotted the test accuracy for k values of 1, 3, 5, 7, 9, 11, 13, 15, and 17. There does not appear to be a significant trend in any direction, maybe a slight downwards trend but this is negligible in my opinion as the above graph does not start at 0.

3. Discuss the main advantages and disadvantages of k-Nearest Neighbour method.

Advantages:

- It does not require any training before making a prediction for a new instance of the test set, which means that new data can be added seamlessly to the test set at any stage, and this will not impact the overall outcomes of the algorithm.
- It is very simple to implement and easy to understand.

Disadvantages:

- kNN needs scaling in order to work effectively. In order for the Euclidean distance measure to be fair, the program needs to scale each feature either through normalisation of the entire dataset or by each feature in the Euclidean distance calculation being divided by the range. This means that extra work is required to generate accurate predictions.
-
- kNN does not work as well when the training dataset gets larger, as more calculations are needed per classification, so the program can become slow if the training dataset is very large.
- kNN also slows down when there are a large number of features for each instance, as it has to do more calculations per training instance, which when coupled with the first disadvantage can significantly slow down the speed of the algorithm.

4. Assuming that you are asked to apply the k-fold cross-validation method for the above problem with k=5, what would you do? State the major steps.

   Steps:

   1. Recombine the training and test datasets into one large one.
   2. Divide the large dataset into k many equal subsets, in this case, 5.
   3. Take four of the subsets to train the algorithm on and test it on the final subset.
   4. Repeat this k times, each time taking a different subset to be the testing one.
   5. Average the classification accuracy across all of the different folds to produce a single estimation. This produces a more realistic accuracy estimation for the algorithm.

5. In the above problem, assuming that there were actually no class labels available. Which method would you use to group the examples in the data set? State the major steps.

   I would use K-means clustering to classify each instance in the dataset.

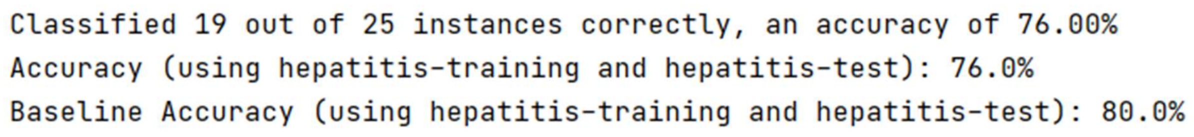
   Steps:

   - Choose how many classes you think there are, k.
   - Generate k initial "means" randomly from the data set.
   - Create k clusters by assigning every instance to the nearest cluster: based on the nearest mean according to the distance measure.
     - This is done by looking at all the centroids in turn and calculating Euclidean distance to work out which is closest.
   - When finished classifying every data point, replace the old means with the centroid (mean) of each new cluster.
   - Repeat the above two steps until convergence (no change in each cluster centroid). This means that the points have all been classified to the best ability of the algorithm.

Part 2: Decision Tree

1. You should first apply your program to the hepatitis-training and hepatitis-test files and report the classification accuracy in terms of the fraction of the test instances that it classified correctly. Report the constructed decision tree classifier printed by your program. Compare the accuracy of your decision tree program to the baseline classifier (which always predicts the most frequent class in the training set), and comment on any difference.

Below is my accuracy for the hepatitis-training and hepatitis-test datasets, for both my decision tree and the baseline classification.



```
Classified 19 out of 25 instances correctly, an accuracy of 76.00%
Accuracy (using hepatitis-training and hepatitis-test): 76.0%
Baseline Accuracy (using hepatitis-training and hepatitis-test): 80.0%
```

When running my decision tree against the baseline, I have a worse accuracy, as the decision tree has correctly classified one less instance than the baseline program has classified. I believe that the reason the baseline accuracy is so good is that the dataset does not contain an even distribution of classes, which skews the tree, as it has less information to make a more accurate split.

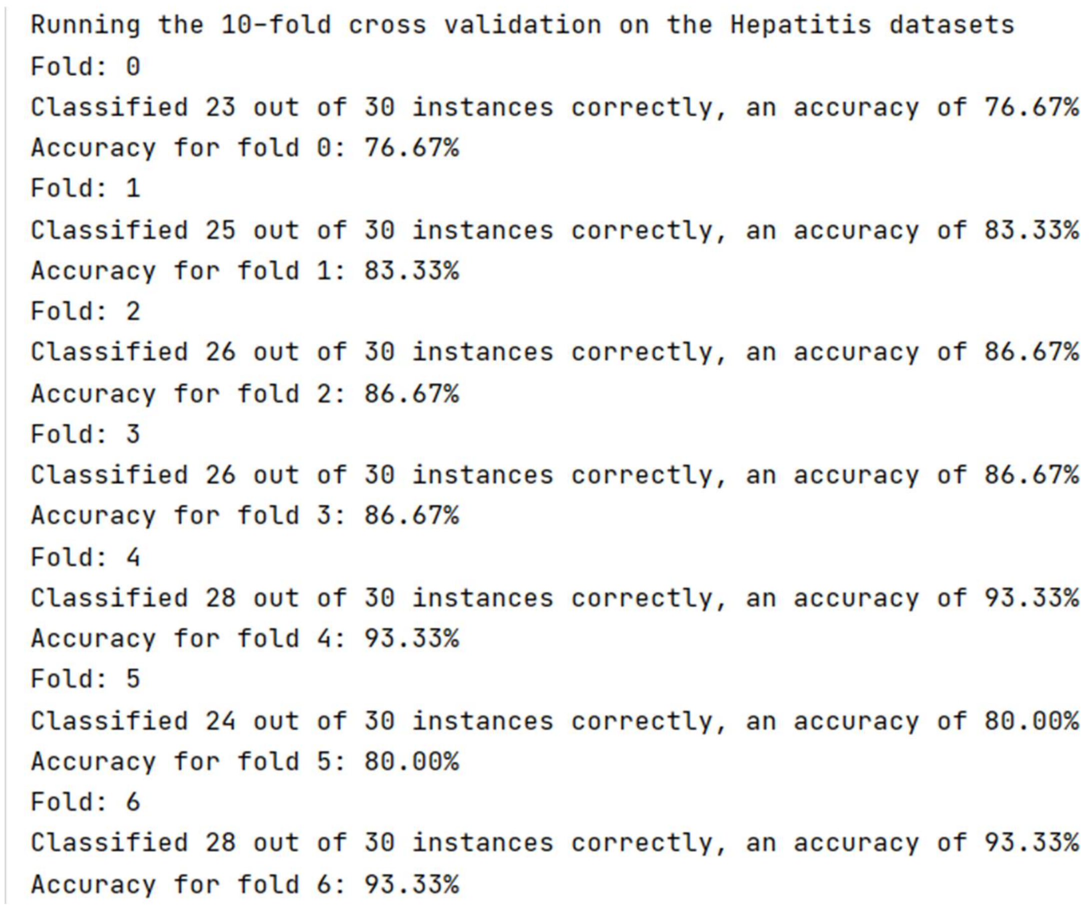
This distribution has a two-fold effect, as an unevenly distributed dataset not only makes my tree worse, but it improves the baseline classification as well, hence the resulting poor relative performance of my tree. My accuracy for the tree is 76%, classifying 19/25 instances correctly, whereas the baseline accuracy is 80%, classifying 20/25 instances correctly. I think this difference is negligible, as it is only one instance, and the accuracy may be different if a different dataset is used.

Below is my tree that is generated by the program. Note that I have not reused attributes as I felt that the tree was already quite long, and I did not see a significant improvement in accuracy when I made it longer.

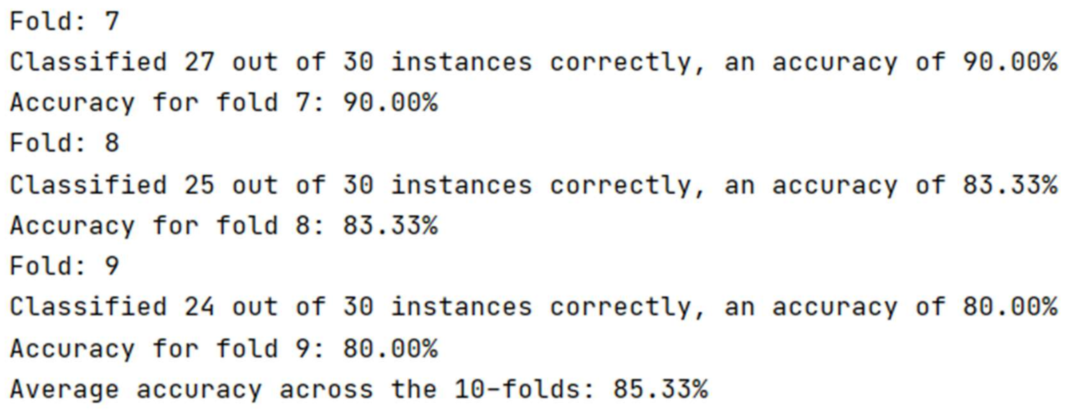
```
Separating Feature: ASCITES
    true node:
        Separating Feature: SPIDERS
            true node:
                Separating Feature: VARICES
                    true node:
                        Separating Feature: STEROID
                            true node:
                                Most Likely Class: live
                                Probability: 1.0
                            false node:
                                Separating Feature: SPLEENPALPABLE
                                    true node:
                                        Separating Feature: FIRMLIVER
                                            true node:
                                                Most Likely Class: live
                                                Probability: 1.0
                                            false node:
                                                Separating Feature: BIGLIVER
                                                    true node:
                                                        Separating Feature: SGOT
                                                            true node:
                                                                Most Likely Class: live
                                                                Probability: 1.0
                                                            false node:
                                                                Separating Feature: FEMALE
                                                                    true node:
                                                                        Most Likely Class: live
                                                                        Probability: 1.0
                                                                    false node:
                                                                        Separating Feature: ANOREXIA
                                                                            true node:
                                                                                Most Likely Class: die
                                                                                Probability: 1.0
                                                                            false node:
                                                                                Most Likely Class: live
                                                                                Probability: 1.0
                                                    false node:
                                                        Most Likely Class: live
                                                        Probability: 1.0
                                    false node:
                                        Separating Feature: HISTOLOGY
                                            true node:
                                                Most Likely Class: die
                                                Probability: 1.0
                                            false node:
                                                Most Likely Class: live
                                                Probability: 1.0
                false node:
                        Most Likely Class: die
                        Probability: 1.0
                false node:
                Separating Feature: BILIRUBIN
                        true node:
                        Separating Feature: FATIGUE
                            true node:
                                Separating Feature: AGE
                                    true node:
                                        Most Likely Class: live
                                        Probability: 1.0
                                    false node:
                                        Most Likely Class: die
                                        Probability: 1.0
                            false node:
                                Separating Feature: ANTIVIRALS
                                    true node:
                                        Separating Feature: MALAISE
                                            true node:
                                                Most Likely Class: live
                                                Probability: 0.75
                                            false node:
                                                Most Likely Class: live
                                                Probability: 0.7
                                    false node:
                                        Most Likely Class: live
                                        Probability: 1.0
                        false node:
                        Most Likely Class: live
                        Probability: 0.8888888888888888
false node:
    Most Likely Class: die
    Probability: 0.7333333333333333
```

2. You should then apply 10-fold cross-validation to evaluate the robustness of your algorithm. We have provided files for the split training and test sets. The files are named as hepatitis-training-run-*, and hepatitis-test-run-*. Each training set has 107 instances, and each test set has the remaining 30 instances. You should train and test your classifier on each pair and calculate the average accuracy of the classifiers across the 10 folds (show your working).

The screenshot included below is my accuracy for each individual set, and then my overall accuracy is included at the bottom, along with my calculation.



```
Running the 10-fold cross validation on the Hepatitis datasets
Fold: 0
Classified 23 out of 30 instances correctly, an accuracy of 76.67%
Accuracy for fold 0: 76.67%
Fold: 1
Classified 25 out of 30 instances correctly, an accuracy of 83.33%
Accuracy for fold 1: 83.33%
Fold: 2
Classified 26 out of 30 instances correctly, an accuracy of 86.67%
Accuracy for fold 2: 86.67%
Fold: 3
Classified 26 out of 30 instances correctly, an accuracy of 86.67%
Accuracy for fold 3: 86.67%
Fold: 4
Classified 28 out of 30 instances correctly, an accuracy of 93.33%
Accuracy for fold 4: 93.33%
Fold: 5
Classified 24 out of 30 instances correctly, an accuracy of 80.00%
Accuracy for fold 5: 80.00%
Fold: 6
Classified 28 out of 30 instances correctly, an accuracy of 93.33%
Accuracy for fold 6: 93.33%
```



```
Fold: 7
Classified 27 out of 30 instances correctly, an accuracy of 90.00%
Accuracy for fold 7: 90.00%
Fold: 8
Classified 25 out of 30 instances correctly, an accuracy of 83.33%
Accuracy for fold 8: 83.33%
Fold: 9
Classified 24 out of 30 instances correctly, an accuracy of 80.00%
Accuracy for fold 9: 80.00%
Average accuracy across the 10-folds: 85.33%
```

Average accuracy = $\frac{(76.67 + 83.33 + 86.67 + 86.67 + 93.33 + 80.00 + 93.33 + 90.00 + 80.00 + 80.00)}{10}$ = 85%

3. "Pruning" (removing) some of the leaves of the decision tree will always make the decision tree less accurate on the training set. Explain: (a) how you could prune leaves from the decision tree; (b) why it reduces accuracy on the training set; and (c) why it might improve accuracy on the test set.

a) Pruning is when we randomly remove a node and its associated subtree from the overall decision tree. This helps generalise the tree as by removing random nodes we make the tree less specific to the training set, therefore more general.

   These are the steps that I would follow:

   1. Initialise a pruning probability and make it very low.
   2. Traverse the tree breadth-first, and for every node generate a random number between 0 and 1 and if it is less than the pruning probability, remove the node from the tree.
   3. As you go down each level, increase the pruning probability, as the chance that the node is hyper-specific and overfitted increases the further down the tree you go. This means that the nodes that get removed will be more likely to come from further down the tree.

b) By definition, a tree that is generalised will never do better than a tree that has been trained on the dataset you want to test on. This is because there will be outlier instances in the data which will not be covered properly in a general tree, but a tree trained on the dataset may make specific paths to cater to the outliers. This is why it reduces accuracy, as the generalised tree does not account for the outliers.

c) The reason a generalised tree does better on the test set is essentially the same reason that an overfitted tree does better on the training set, as it doesn't account for the odd outliers of the training set, so will be more accurate in the general scenarios. A pruned tree will be less overfitted to the training data therefore will do better on the test set.

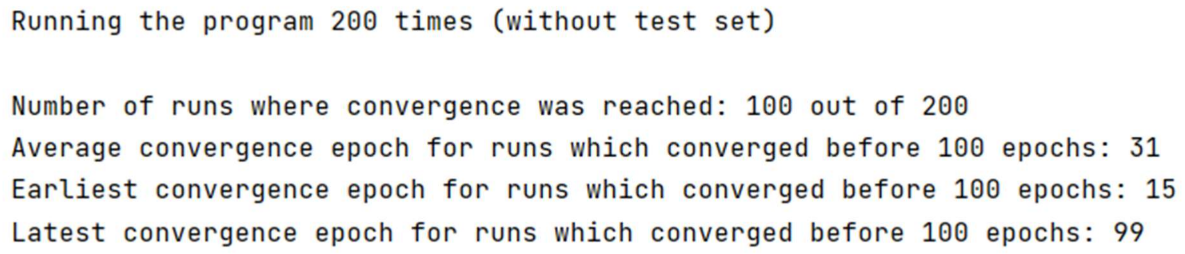
4. Explain why the impurity measure (from lectures) is not an appropriate measure to use if there are three or more classes in the dataset.

- If we have two classes, and a subset of the tree contains 10 nodes of class A and 0 nodes of class B, when we multiply out the impurities, we will get 0, as when we multiply by 0, we get 0, and this will be correct, as there is only one type of class in the subset.
- If, however, we have 3 classes, and a subset of the tree contains 5 nodes of class A, 5 nodes of class B, and 0 nodes of class C when we calculate the impurities when we multiply by the 0 instances of C, we will end up with an impurity of 0. This is not true, however, as there are multiple instances of both class A and B in the subset, so impurity is not an appropriate measure when the number of classes in the dataset is 3 or more.

Part 3: Perceptron

1. Report on the accuracy of your perceptron. For example, did it find a correct set of weights? Did its performance change much between different runs?

   - When the perceptron finds a set of weights which lead it to predict every instances' class correctly we say that the perceptron has 'converged', that is it has reached a point where it no longer needs to train as it has generated a correct set of weights.
   - From preliminary testing I noticed that the performance would vary wildly depending on how good the initial features that were generated tended to be. If I got lucky, it was converging around 20-30 epochs in, if the set of generated features was bad it could run for 100 epochs without reaching convergence.
   - I decided on several performance metrics which I explored and analysed below
   - To analyse what was going on I created a method to run the program for 200 runs, keeping track of how often it converged, and these are the results:



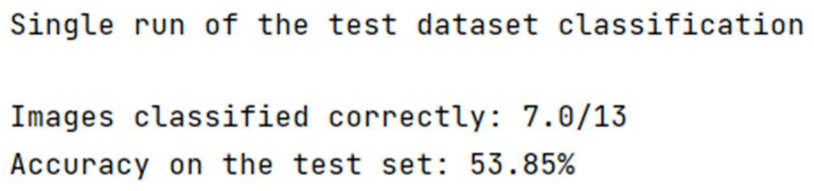
```
Running the program 200 times (without test set)

Number of runs where convergence was reached: 100 out of 200
Average convergence epoch for runs which converged before 100 epochs: 31
Earliest convergence epoch for runs which converged before 100 epochs: 15
Latest convergence epoch for runs which converged before 100 epochs: 99
```

   - From this data we can see that it converges roughly half the time, and the convergence epoch can vary wildly from run to run, with the most extreme values seen being 18 epochs on the quicker end, and 99 epochs on the slower end.

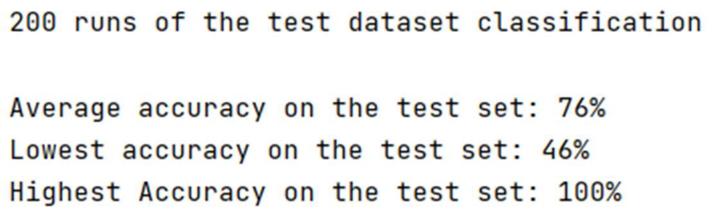
2. Explain why evaluating the perceptron's performance on the training data is not a good measure of its effectiveness. For an A+, you should create additional data to get a better measure (e.g., using MakeImage.java). If you do, report on the perceptron's performance on this additional data.

   - The main reason that evaluating the performance on the training dataset is not a good idea is because we have no way to tell whether or not the data is overfitted. This means that we have no way of knowing what the performance of the perceptron would be like on an unseen dataset, whether it would generalise enough in order to generate a good prediction.
   - Using the provided helper code, I built several different Xs' and Os', combined them into a single 'test-images.data' file, and created a classify test images function. I then ran the perceptron 200 times using this data, to calculate an average accuracy.
   - My results are shown below:



```
Single run of the test dataset classification

Images classified correctly: 7.0/13
Accuracy on the test set: 53.85%
```



```
200 runs of the test dataset classification

Average accuracy on the test set: 76%
Lowest accuracy on the test set: 46%
Highest Accuracy on the test set: 100%
```

   - The results generated by running the classification algorithm 200 times vary wildly again just like the convergence epochs in part one of this question. The average accuracy is reasonable, not incredibly high but definitely a lot better than the lowest accuracy on the 200 runs, which came in as worse than pure guesswork. The highest accuracy was also very good.